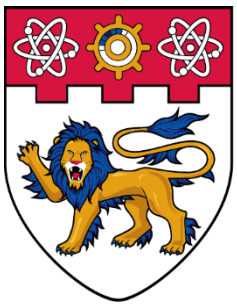


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4042
NEURAL NETWORKS & DEEP LEARNING
ASSIGNMENT 2

KOH TAT YOU @ ARTHUR KOH
(U1821604B)

Contents

PART A: OBJECT RECOGNITION

In Part A, we are tasked to design an SGD CNN to perform object recognition on the CIFAR-10 dataset. The designed CNN model layer specification details are as follow:

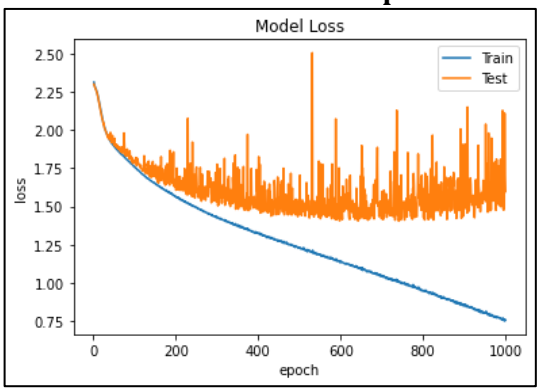
Input	32x32x3 dimensions
Convolution C_1	50 channels window size 9x9 VALID padding ReLU activation
Max Pooling S_1	pooling window size 2x2 stride = 2 VALID padding
Convolution C_2	60 channels window size 5x5 VALID padding ReLU activation
Max Pooling S_2	pooling window size 2x2 stride = 2 VALID padding
Fully Connected F_3	size 300 no activation
Fully Connected F_4	size 10

```
model = tf.keras.Sequential()

# Input Layer
model.add(layers.Input(shape=(3072,)))
model.add(layers.Reshape(target_shape=(32, 32, 3), input_shape=(3072,)))

# C₁ Convolutional Layer
model.add(layers.Conv2D(num_ch_c1, 9, padding='valid', activation='relu', input_shape=(N

# S₁ Max Pooling Layer
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2), padding='valid'))

# C₂ Convolutional Layer
model.add(layers.Conv2D(num_ch_c2, 5, padding='valid', activation='relu', input_shape=(N

# S₂ Max Pooling Layer
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2), padding='valid'))

# Flatten
model.add(layers.Flatten())

# F₃ Fully-Connected Layer
model.add(layers.Dense(300, use_bias=True))  # Here no softmax because we have combined

# F₄ Fully-Connected Layer
model.add(layers.Dense(10, use_bias=True, input_shape=(300,)))  # Here no softmax becaus
```

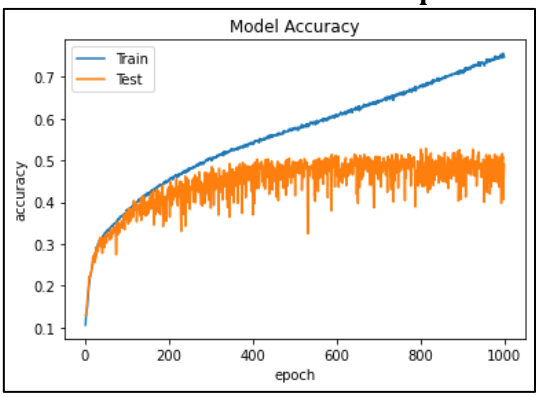
Question 1a

We are tasked to use our previously designed SGD CNN and plot the training cost, test cost, training accuracy and test accuracy against learning epochs. They are graphed below as follow:

Train & Test Costs vs Epochs



Train & Test Accuracies vs Epochs



From the graphs above, we observe the following phenomena with our CNN as noted below:

- Test Loss Convergence
  - Occurs: ~420 epochs
  - Loss MSE: ~1.45
- Train Loss continuously decreases, in an almost linear fashion.
- Test Loss decreases slightly, until ~825 epochs, increasing thereafter.

- Test Accuracy Convergence
  - Occurs: ~420 epochs
  - Accuracy: ~0.49
- Train Accuracy continuously increases, in an almost linear fashion.
- Test Accuracy fluctuates, but remains relatively within the range of ~0.50.

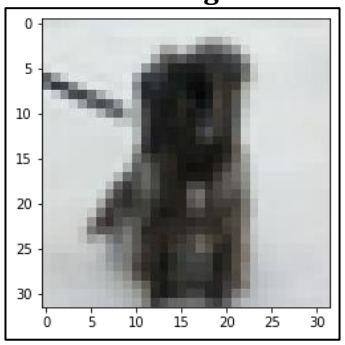
From the above, we postulate that our CNN model is overfitted on the training data. We also note that we are using almost twice the number of epochs needed for model convergence to occur.

Question 1b

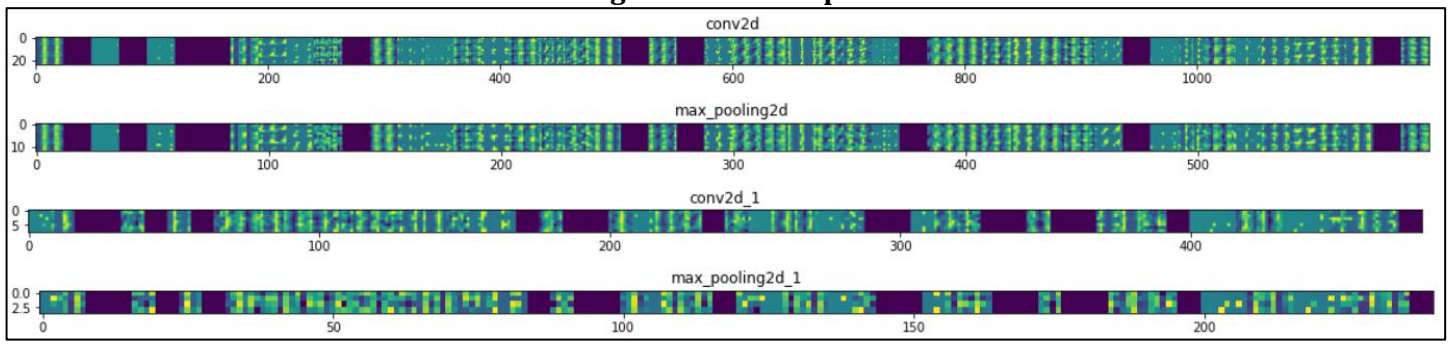
We are tasked to show 2 test images, along with their corresponding plot feature maps at convolutional layers and pooling layers within our CNN.

Test Image 1 and its corresponding feature map summary are pictured below as so:

Test Image 1

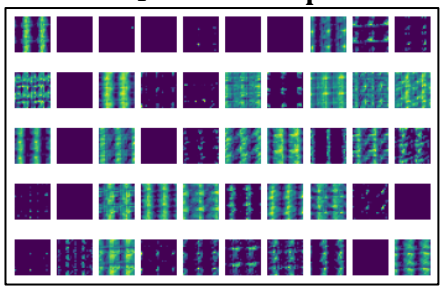


Test Image 1 Feature Map Overview

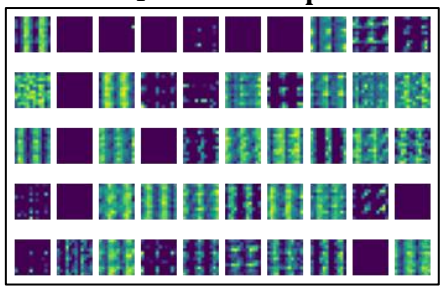


The discrete feature map showing Test Image 1's individual features for each respective layer are pictured below as so:

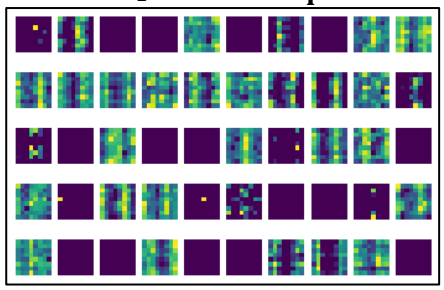
C_1 Feature Map



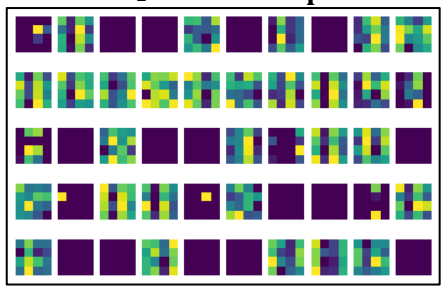
S_1 Feature Map



C_2 Feature Map

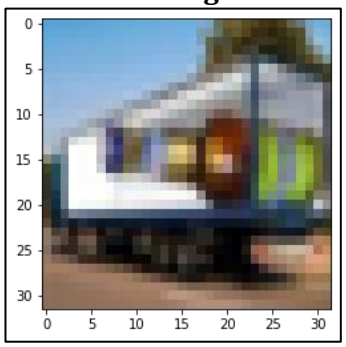


S_2 Feature Map

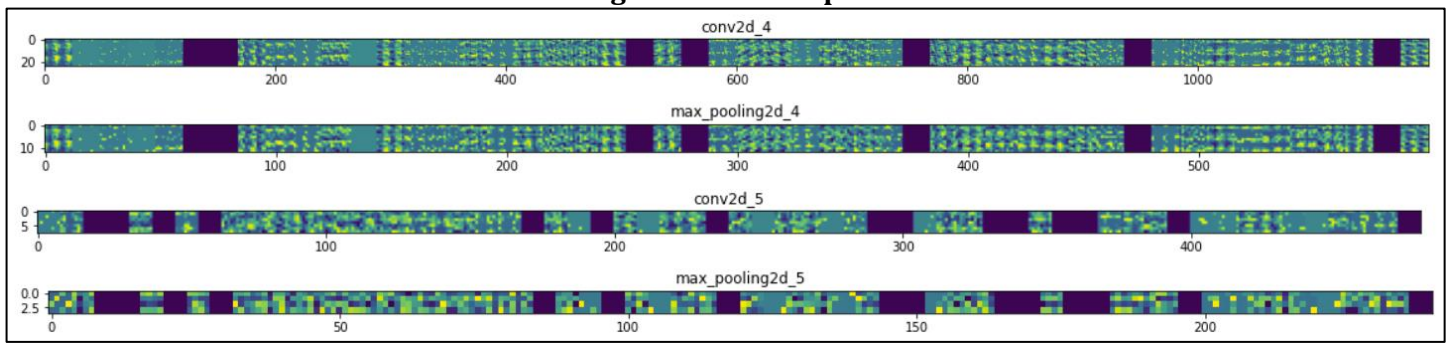


Test Image 2 and its corresponding feature map summary are pictured below as so:

Test Image 2

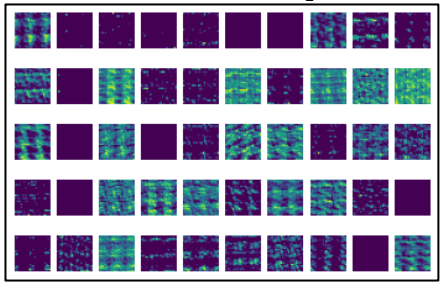


Test Image 2 Feature Map Overview

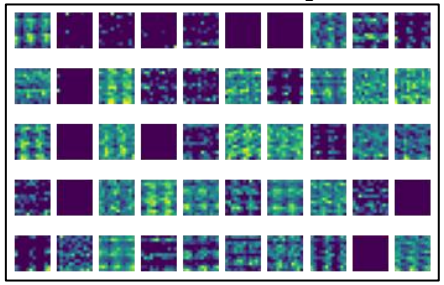


The discrete feature map showing Test Image 2's individual features for each respective layer are pictured below as so:

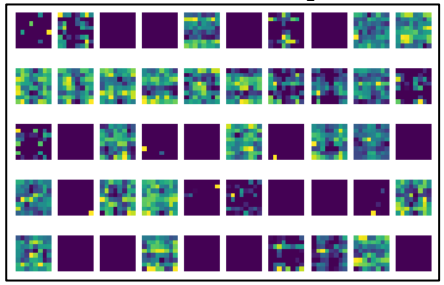
C_1 Feature Map



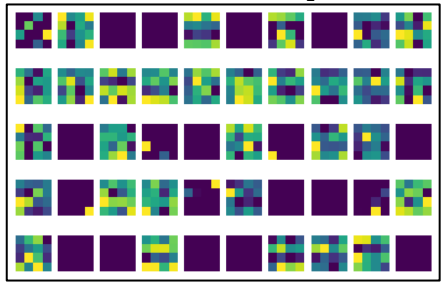
S_1 Feature Map



C_2 Feature Map



S_2 Feature Map



Question 2

We are tasked to use a grid search to discover the optimal combination of convolutional channel pair values for use with our previously designed CNN. The accuracy values corresponding to all 25 channel combinations are reported below as so:

Using Max Validation Accuracy

C_1	C_2	*Max Val Acc*
90	100	0.5425
90	80	0.5335
70	60	0.5330
70	100	0.5315
50	100	0.5305
90	40	0.5305
50	60	0.5300
30	100	0.5285
70	80	0.5280
70	20	0.5260
50	80	0.5245
90	60	0.5230
30	80	0.5220
50	40	0.5205
10	80	0.5160
10	100	0.5160
70	40	0.5150
30	60	0.5145
30	40	0.5145
10	40	0.5140
50	20	0.5135
90	20	0.5120
30	20	0.5110
10	60	0.5105
10	20	0.4945

Using Avg Validation Accuracy

C_1	C_2	*Avg Val Acc*
90	100	0.4892
30	100	0.4869
70	100	0.4862
70	80	0.4861
50	100	0.4827
70	60	0.4809
70	20	0.4794
50	20	0.4787
50	40	0.4769
10	40	0.4763
10	100	0.4746
70	40	0.4742
90	80	0.4737
50	80	0.4733
90	20	0.4731
50	60	0.4728
10	80	0.4726
90	40	0.4710
30	40	0.4696
90	60	0.4693
30	80	0.4678
10	60	0.4674
30	20	0.4645
30	60	0.4612
10	20	0.4449

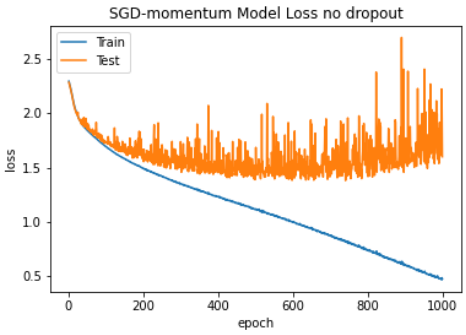
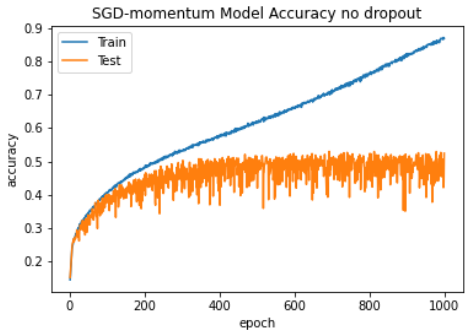
From the both sets of results tabled above, we observe that the optimal channel combination numbers are [C_1, C_2] = [*90*, *100*], as both the Max Validation Accuracy and Average Validation Accuracy values are the highest in comparison to the other channel combination respective accuracy values.

Question 3

We are tasked to use the previously obtained optimal combination, [90, 100] to train 4 CNN variants to perform object recognition on the CIFAR-10 dataset and plot the costs and accuracies against epochs.

Question 3a

We trained an SGD-Momentum CNN variant with momentum $\gamma = 0.1$ to perform object recognition on the CIFAR-10 dataset. The cost and accuracies against epochs plots are graphed below as such:

Train & Test Costs vs Epochs	Train & Test Accuracies vs Epochs
	

From the graphs above, we observe the following phenomena with our SGD-Momentum CNN variant as noted below:

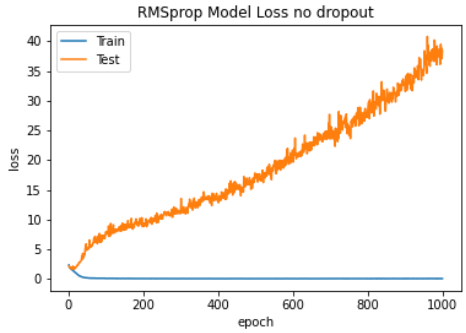
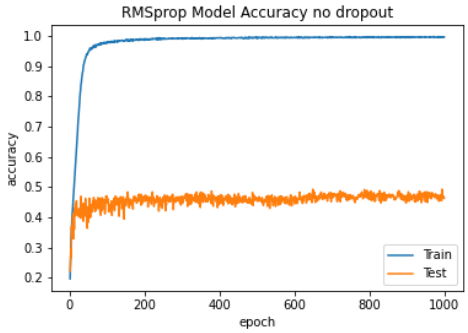
- Test Loss Convergence
  - Occurs: ~420 epochs
  - Loss MSE: ~1.45
- Train Loss continuously decreases, in an almost linear fashion.
- Test Loss decreases slightly, until ~825 epochs, increasing thereafter.

- Test Accuracy Convergence
  - Occurs: ~420 epochs
  - Accuracy: ~0.49
- Train Accuracy continuously increases almost linearly, with a slight curvature
- Test Accuracy fluctuates, but remains relatively within the range of ~0.50.

From the above, we observe that the use of momentum has no profound change on the performance of our initially defined CNN model. As such, we postulate that this model also shares the same issue as the model it is based on – in that it is highly likely that it is being overfitted on the training data used. As before, we also note that this model converges at around the same point as the model it is based on.

Question 3b

We trained an RMSProp CNN variant to perform object recognition on the CIFAR-10 dataset. The cost and accuracies against epochs plots are graphed below as such:

Train & Test Costs vs Epochs	Train & Test Accuracies vs Epochs
	

From the graphs above, we observe the following phenomena with our RMSProp CNN variant as noted below:

- Test Loss Convergence
  - Occurs: - (no occurrence)
  - Loss MSE: ~0
- Train Loss linearly decreases at the start, before flatlining at 0.
- Test Loss increases without end with a small increase in variance.

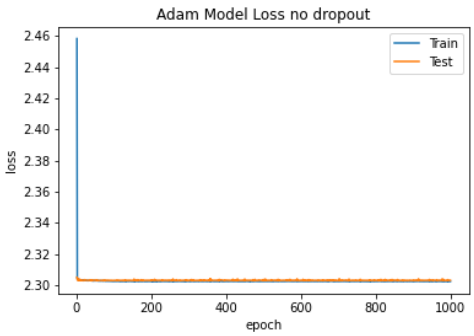
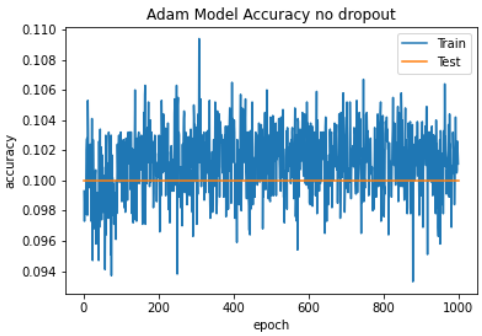
- Test Accuracy Convergence
  - Occurs: ~25 epochs
  - Accuracy: ~0.47
- Train Accuracy increases extremely quickly before remaining at ~0.95
- Test Accuracy increases extremely quickly before remaining at ~0.42 and gradually decreasing in variance

From the above, we observe that the use of RMSProp drastically alters the performance of our initially defined CNN model. In particular, the model now learns extremely quickly from the input dataset. Unfortunately, this extreme high accuracy also points to the idea that this model variant is overfitted on the data used for training it. As such, it is not able to generalize well, thus giving rise to poorer test accuracy values, as compared to the previous models.

We also note that this model fails to converge in terms of its loss metric values, and that while it converges early in terms of accuracy metric values, it due to overfitting, and as such, is indicative of a good model for object recognition.

Question 3c

We trained an Adam optimizer CNN variant to perform object recognition on the CIFAR-10 dataset. The cost and accuracies against epochs plots are graphed below as such:

Train & Test Costs vs Epochs	Train & Test Accuracies vs Epochs
	

From the graphs above, we observe the following phenomena with our Adam CNN variant as noted below:

- Test Loss Convergence
  - Occurs: ~1 epoch
  - Loss MSE: ~2.30
- Train Loss spikes at the first epoch to 2.46, before immediately flatlining to ~2.30 and remaining around that value with little variance.
- Test Loss immediately starts at ~2.30 and remaining around that value little variance.

- Test Accuracy Convergence
  - Occurs: ~0 epoch
  - Accuracy: 0.1
- Train Accuracy varies little, starting at 0.098 and remaining around that value with little variance (seems large due to scale of diagram, but the variance is actually very small).
- Test Accuracy remains at a constant 0.1 from the beginning to the end.

From the above, we observe that the use of the Adam optimizer drastically alters the performance of our initially defined CNN model. The model, in its current state is not useful as it fails to learn, and in fact, cannot even overfit on the input data.

We also note that while this model variant converges very early on, at 0 epoch in terms of loss, and 1 epoch in terms of accuracy, is not indicative of any useful object recognition capability.

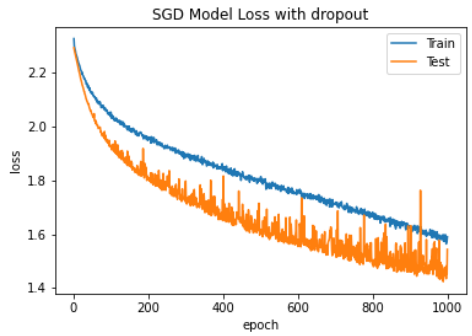
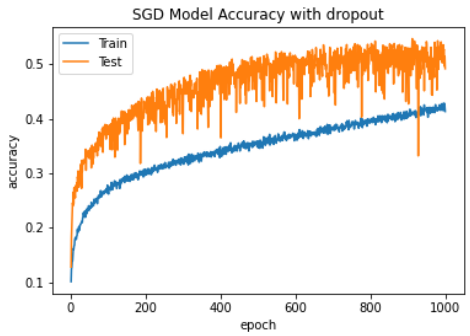
Question 3d

We trained an SGD CNN (with dropout) variant to perform object recognition on the CIFAR-10 dataset. Due to the addition of dropout, the layer specifications for this model is slightly tweaked, and is detailed below as follow:

Input	32x32x3 dimensions
Convolution C_1	50 channels window size 9x9 VALID padding ReLU activation
Max Pooling S_1	pooling window size 2x2 stride = 2 VALID padding
Convolution C_2	60 channels window size 5x5 VALID padding ReLU activation
Max Pooling S_2	pooling window size 2x2 stride = 2 VALID padding
Fully Connected F_3	size 300 no activation
Dropout D_1	Dropout = 0.5
Fully Connected F_4	size 10
Dropout D_2	Dropout = 0.5

```
# Input Layer
model.add(layers.Input(shape=(3072,)))
model.add(layers.Reshape(target_shape=(32, 32, 3), input_sha
# C₁ Convolutional Layer
model.add(layers.Conv2D(num_ch_c1, 9, padding='valid', activ
# S₁ Max Pooling Layer
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2),
# C₂ Convolutional Layer
model.add(layers.Conv2D(num_ch_c2, 5, padding='valid', activ
# S₂ Max Pooling Layer
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2),
# Flatten
model.add(layers.Flatten())
# F₃ Fully-Connected Layer
model.add(layers.Dense(300, use_bias=True))  # Here no softm
# D₁ Dropout Layer
model.add(layers.Dropout(0.5))
# F₄ Fully-Connected Layer
model.add(layers.Dense(10, use_bias=True, input_shape=(10,))
# D₂ Dropout Layer
model.add(layers.Dropout(0.5))
```

The cost and accuracies against epochs plots are graphed below as such:

Train & Test Costs vs Epochs	Train & Test Accuracies vs Epochs
	

From the graphs above, we observe the following phenomena with our SGD CNN (with dropout) variant as noted below:

- Test Loss Convergence
  - Occurs: - (no occurrence)
  - Loss MSE: - (no occurrence)
- Train Loss decreases steeply before the first 150 epochs, before decreasing less steeply (in an almost linear fashion) with small variance

- Test Accuracy Convergence
  - Occurs: ~750 epochs
  - Accuracy: ~0.51
- Train Accuracy increases sharply at the beginning before the first 150 epochs, before increasing more gently (in an almost linear fashion) with small variance.

- Test Loss decreases steeply before the first 150 epochs, before decreasing less steeply with spikes in the loss value.
- Test Accuracy increases sharply at the beginning before the first 150 epochs, before increasing more gently (in an almost linear fashion) with negative spikes in the accuracy value.

From the above, we observe that the use of dropout seemingly results in a slight performance increase of our initially defined CNN model, although we were not able to observe loss convergence due to limited number of epochs used for training.

Unfortunately, while better, we still postulate that the likelihood that this CNN model variant is still being overfitted is high. We also note that this model, while higher in accuracy, takes a longer time to converge, at around 750 epochs. Before convergence, its accuracy performance takes around the same amount of time as the initial model it was based on. This mean that the ceiling for this model is higher than other models, although there is an element of diminishing returns,

Question 4

We compare the accuracies of all the models in the table below:

Model Variant	**Type**	**C_1**	**C_2**	**Max Validation Acc**	**Mean Validation Acc**
SGD Model	Base	50	60	0.5289	0.4440
SGD Model	Optimal	90	100	0.5425	0.4496
SGD-Momentum	Optimal	90	100	0.5304	0.4534
RMSProp	Optimal	90	100	0.4934	0.4585
Adam Optimizer	Optimal	90	100	0.1000	0.1000
SGD w/ Dropout	Optimal	90	100	0.5465	0.4598

From the table above, we observe that the *best performing* model is the *SGD w/ Dropout* CNN variant, which has the highest Max and Mean Validation Accuracy values.

We also observe that the *worst performing* model is the *Adam Optimizer* CNN variant, which has the lowest Max and Mean Validation Accuracy values, as compared to the other models.

We observe that the SGD Model performs similarly in terms of validation accuracy, with the number of channels making minimal difference. Most of the large changes in model performance come from the use the Model variants. In particular, the Adam Optimizer CNN variant has the most unusual performance metric, with the lowest Max and Mean Validation accuracy values of all the models. We postulate that there is a limiting factor that prevents the Adam Optimizer from properly learning from the dataset, and that the use of dropout might actually improve its performance.

PART B: TEXT CLASSIFICATON

In Part B, we are tasked to implement CNN and RNN layers at the character and word levels for text classification of paragraphs collected from Wikipage entries.

The designed Character CNN model layer specification details are as follow:

Input	32x32x3 dimensions
Convolution C_1	10 filters Window size 20x256 VALID padding ReLU activation
Max Pooling S_1	pooling window size 4x4 stride = 2 padding = 'SAME'
Convolution C_2	10 filters Window size 20x1 VALID padding ReLU activation
Max Pooling S_2	pooling window size 4x4 stride = 2 padding = 'SAME'
Fully Connected F_3	size 15 Softmax activation

```
# Designing of CharCNN model
tf.keras.backend.set_floatx('float32')
class CharCNN(Model):
    def __init__(self, vocab_size=256):
        super(CharCNN, self).__init__()
        self.vocab_size = vocab_size
        #Weight variables and CNN cell
        self.conv1 = layers.Conv2D(N_FILTERS, FILTER_SHAPE1, padding='VALID
        self.pool1 = layers.MaxPool2D(POOLING_WINDOW, POOLING_STRIDE, paddi
        self.conv2 = layers.Conv2D(N_FILTERS, FILTER_SHAPE2, padding='VALID
        self.pool2 = layers.MaxPool2D(POOLING_WINDOW, POOLING_STRIDE, paddi
        self.flatten = layers.Flatten()
        self.dense = layers.Dense(MAX_LABEL, activation='softmax')

    def call(self, x, drop_rate=0.5):
        #forward
        x = tf.one_hot(x, one_hot_size)
        x = x[..., tf.newaxis]
        x = self.conv1(x)
        x = self.pool1(x)
        x = self.conv2(x)
        x = self.pool2(x)
        x = self.flatten(x)
        x = tf.nn.dropout(x, drop_rate)
        logits = self.dense(x)
        return logits
```

The designed Word CNN model layer specification details are as follow:

Input	32x32x3 dimensions
Convolution C_1	10 filters Window size 20x256 VALID padding ReLU activation
Max Pooling S_1	pooling window size 4x4 stride = 2 padding = 'SAME'
Convolution C_2	10 filters Window size 20x1 VALID padding ReLU activation
Max Pooling S_2	pooling window size 4x4 stride = 2 padding = 'SAME'
Fully Connected F_3	size 15 Softmax activation

```
class WordCNN(Model):
    def __init__(self, vocab_size=256):
        super(WordCNN, self).__init__()
        self.vocab_size = vocab_size
        #Weight variables, embedding layer & CNN cell
        self.embedding = layers.Embedding(vocab_size, EMBEDDING_SIZE, input_
        self.conv1 = layers.Conv2D(N_FILTERS, FILTER_SHAPE3, padding='VALID
        self.pool1 = layers.MaxPool2D(POOLING_WINDOW, POOLING_STRIDE, paddi
        self.conv2 = layers.Conv2D(N_FILTERS, FILTER_SHAPE2, padding='VALID
        self.pool2 = layers.MaxPool2D(POOLING_WINDOW, POOLING_STRIDE, paddi
        self.flatten = layers.Flatten()
        self.dense = layers.Dense(MAX_LABEL, activation='softmax')

    def call(self, x, drop_rate=0.5):
        #forward
        x = self.embedding(x)
        x = x[..., tf.newaxis]
        x = self.conv1(x)
        x = self.pool1(x)
        x = self.conv2(x)
        x = self.pool2(x)
        x = self.flatten(x)
        x = tf.nn.dropout(x, drop_rate)
        logits = self.dense(x)
        return logits
```

The designed Char RNN model layer specification details are as follow:

RNN	GRU Layer
Hidden Layer	Size 20 (not pictured)
Embedding Layer	Size 20 (not pictured)

```
class CharRNN(Model):

    def __init__(self, vocab_size=256, hidden_dim=10):
        super(CharRNN, self).__init__()
        self.vocab_size = vocab_size
        self.hidden_dim = hidden_dim

        # Weight variables and RNN cell
        self.rnn = layers.RNN(tf.keras.layers.GRUCell(sel
        self.dense = layers.Dense(MAX_LABEL, activation=N

    def call(self, x, drop_rate):
        # forward logic
        categorical_labels = to_categorical(x, self.vocab
        encoding = self.rnn(categorical_labels)
        encoding = tf.nn.dropout(encoding, drop_rate)
        logits = self.dense(encoding)

        return logits
```

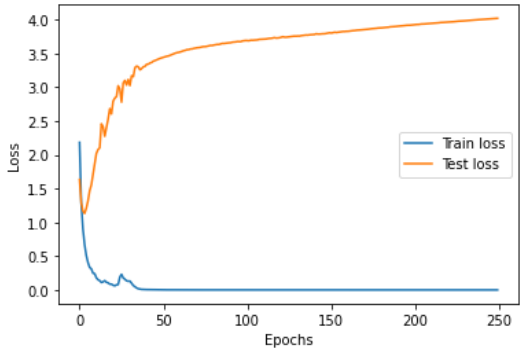
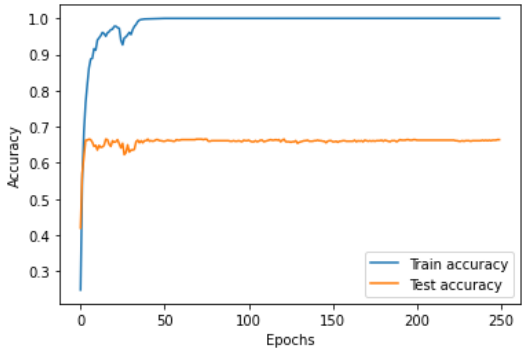
The designed Word RNN model layer specification details are as follow:

RNN	GRU Layer
Hidden Layer	Size 20 (not pictured)
Embedding Layer	Size 20 (not pictured)

```
class WordRNN(Model):

    def __init__(self, vocab_size, hidden_dim=10):
        super(WordRNN, self).__init__()
        # Hyperparameters
        self.hidden_dim = hidden_dim
        self.vocab_size = vocab_size
        self.embedding = layers.Embedding(vocab_size, EMBEDDING

        # Weight variables and RNN cell
        self.rnn = layers.RNN(tf.keras.layers.GRUCell(self.hidd
        self.dense = layers.Dense(MAX_LABEL, activation=None)

    def call(self, x, drop_rate):
        # forward logic
        embedding = self.embedding(x)
        encoding = self.rnn(embedding)
        encoding = tf.nn.dropout(encoding, drop_rate)
        logits = self.dense(encoding)

        return logits
```

Question 1

We trained a Character CNN Classifier on the paragraph dataset. The cost and accuracies against epochs plots are graphed below as such:

Train & Test Costs vs Epochs	Train & Test Accuracies vs Epochs
	

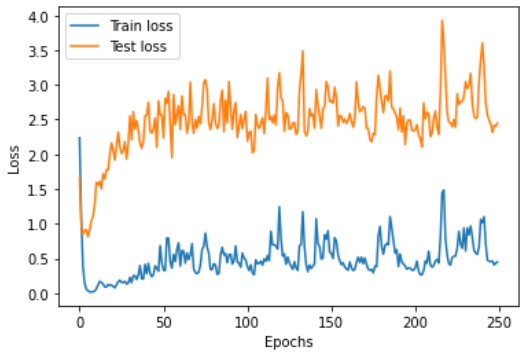
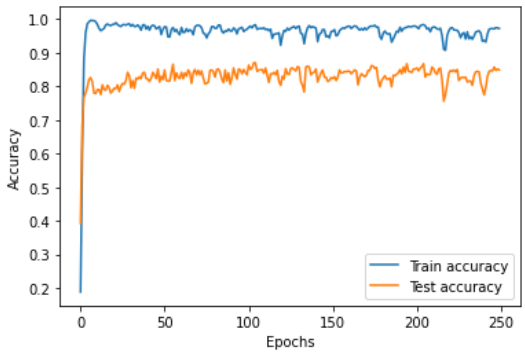
From the graphs above, we observe the following phenomena with our Char CNN Classifier as noted below:

- Test Loss Convergence
  - Occurs: ~240 epochs
  - Loss MSE: ~4.00
- Train Loss immediately starts at 2.25 before spiking negatively at the first epoch to ~0.1. Then, it increases to ~0.25 from epochs 20 to 30 before decreasing to ~0.1 and remaining flatlined for the rest of the epochs.
- Test Loss immediately starts at ~1.6 and immediately plunges to ~1.0 before rapidly increasing to 3.25 during the first 50 epoch cycles. It then gently increases to 4.0 from epochs 50 onwards, for the rest of the epochs.
- Test Accuracy Convergence
  - Occurs: ~35 epochs
  - Accuracy: 0.65
- Train Accuracy starts at 0, then immediately spikes to 0.9 within the first few epochs. It then increases more gently to 0.97 before suffering a sudden dip at around epoch 25 to 0.9. It then steadily increases to 0.99 and remains around there for the rest of the epochs.
- Test Accuracy starts at 0.42, and then spikes to 0.65 in the first few epochs, before it varies up and down until epoch 25, and then it remains relatively stable, at 0.66 from epoch 35 onwards.

From the above, we observe that Char CNN Classifier is most likely overfitting on its training data, as evident from its high train accuracy and low test accuracy. This means that is not able to generalize effectively, and hence identify the 15 categorical classes.

Question 2

We trained a Word CNN Classifier on the paragraph dataset. The cost and accuracies against epochs plots are graphed below as such:

Train & Test Costs vs Epochs	Train & Test Accuracies vs Epochs
	

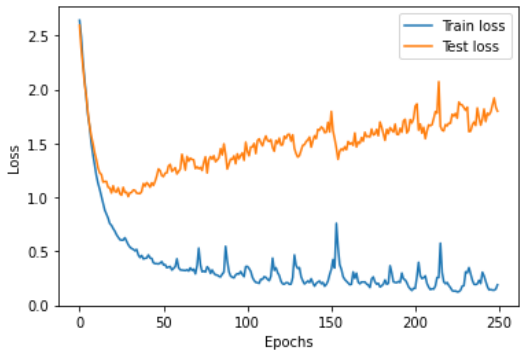
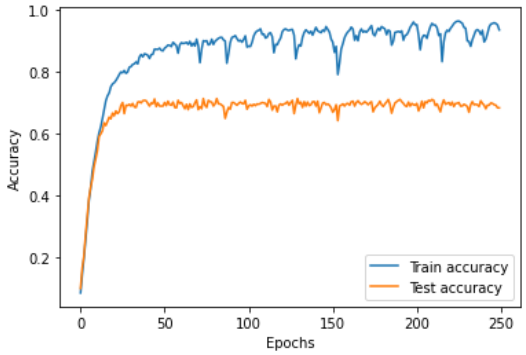
- Test Loss Convergence
  - Occurs: - (no occurrence)
  - Loss MSE: -
- Train Loss starts at 2.25 before immediately spiking negatively in the first few epochs to ~0.1. Then, it increases with variances and large spikes for the rest of the epochs.
- Test Loss immediately starts at ~1.75 and immediately plunges to ~0.9 before rapidly increasing to 1.5 during the first 10 epoch cycles. It then hovers around 2.25 with large variance and massive spikes for the rest of the epochs.

- Test Accuracy Convergence
  - Occurs: ~50 epochs
  - Accuracy: 0.80
- Train Accuracy starts at 0, then immediately spikes to 0.99 within the first few epochs. It then dips slightly to 0.97 within the first 10 epochs, and then slightly decreases to 0.95 with some variance and occasional negative spikes for the rest of the epochs.
- Test Accuracy starts at 0.40, and then immediately spikes to 0.79 in the first epochs, before it increases slightly more gently to 0.83 in the first 10 epochs. It then dips to 0.79 and then slowly increases and hovers around 0.82 with variances until epoch 100. For the rest of epochs, it hovers around there with occasional negative spikes.

From the above, we observe that Word CNN Classifier might be generalizing properly, instead of overfitting like our previous Char CNN Classifier. This is evident from its test accuracy following closely behind its train accuracy. As such, this model is likely to be able to identify the 15 categorical classes.

Question 3

We trained a Character RNN Classifier on the paragraph dataset. The cost and accuracies against epochs plots are graphed below as such:

Train & Test Costs vs Epochs	Train & Test Accuracies vs Epochs
	

- Test Loss Convergence
  - Occurs: - (no occurrence)
  - Loss MSE: -

- Train Loss starts at 2.6 before steeply decreasing to 0.6 in the first 25 epochs. It then decreases less steeply to 0.4 until epoch 50. Following that, it slowly decreases with occasional large spikes for the rest of the epochs.

- Test Loss starts at 2.6 before steeply decreasing to 1.2 in the first few epochs. It then decreases less steeply until to 1.0 until epoch 35, where it starts to increase with variance for the rest of the epochs. To note, there are occasional positive and negative spikes as the training progresses.

- Test Accuracy Convergence
  - Occurs: ~25 epochs
  - Accuracy: 0.90

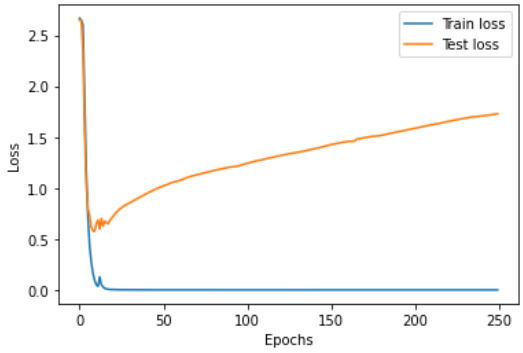
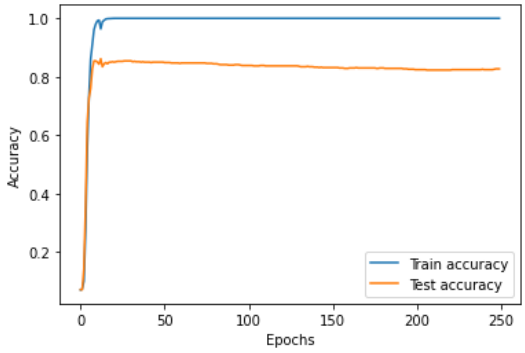
- Train Accuracy starts at 0, then immediately spikes to 0.8 within the first few epochs. It then gently increases to 0.89 until epoch 50. Then, it gradually increases to 0.9 for the rest of the epochs with occasional massive negative spikes.

- Test Accuracy starts at 0, then immediately spikes to 0.6 within the first few epochs. It then gently increases to 0.68 until epoch 25. Then, it hovers around there with small variances for the rest of the epochs.

From the above, we observe that Char RNN Classifier is not performing very well, as its ability to generalize and hence identify the 15 categorical classes is poor.

Question 4

We trained a Word RNN Classifier on the paragraph dataset. The cost and accuracies against epochs plots are graphed below as such:

Train & Test Costs vs Epochs	Train & Test Accuracies vs Epochs
	

- Test Loss Convergence
  - Occurs: 250 epochs
  - Loss MSE: 1.6
- Train Loss starts at 2.6 before steeply decreasing to 0.1 in the first 10 epochs. It then increases a bit for a few epochs, before decreasing by epoch 20, and then remaining flatlined near 0.0 for the rest of the epochs.
- Test Loss starts at 2.6 before steeply decreasing to 0.6 in the first 10 epochs. It then decreases less steeply for a few epochs before varying up and down until epoch 20. Following that, it gently increases to 1.6 for the rest of the epochs.

- Test Accuracy Convergence
  - Occurs: ~25 epochs
  - Accuracy: 0.90
- Train Accuracy starts at 0.05 and then immediately spikes to 0.99 within the first few epochs. It dips for a few epochs to 0.98 and then goes back up to 0.99 and remains there for the rest of the epochs.
- Test Accuracy starts at 0.05 and then immediately spikes to 0.85 within the first few epochs. It dips for a few epochs to 0.83 and then very gently decreases to 0.82 for the rest of the epochs.

From the above, we observe that the Word RNN Classifier is able to perform well, as evident from its test accuracy following closely behind its train accuracy. As such, this model is likely to be able to identify the 15 categorical classes.

Question 5

We compare the accuracies and training time of all the models (without and with dropout) in the table below:

Without Dropout			
Model	Max Validation Acc	Mean Validation Acc	Time (seconds)
CharCNN	0.6814	0.6698	139
WordCNN	0.8471	0.8132	317
CharRNN	0.7142	0.6769	2583
WordRNN	0.8614	0.8239	2492

With Dropout			
Model	Max Validation Acc	Mean Validation Acc	Time (seconds)
CharCNN	0.6542	0.6210	129
WordCNN	0.7914	0.7487	257
CharRNN	0.7142	0.6567	2163
WordRNN	0.8842	0.8657	2300

From the results tabled above, we postulate that for Text Classification the use of dropout:

- RNN models, while around 5-10% percent better than their CNN counterparts for Text Classification, are very computationally expensive.
- To put into perspective, a Char RNN (without dropout) has around 1% increased mean validation accuracy as compared to its Char CNN (without dropout) counterpart. However, the training duration is increased by a factor of almost 18x.
- A Word RNN has (without dropout) has around 1% increased mean validation accuracy as compared to its Char CNN (without dropout) counterpart. However, the training duration is increased by a factor of almost 7x.
- Using dropout results in a definite decreased training time for all models CNN or RNN.
- Model accuracy is more varied with the use of dropout. It has been shown that using dropout results in a decrease of mean validation accuracy for Char CNN, Word CNN & Char RNN.
- However, there is a 5% increase in mean validation accuracy for Word RNN, which is interesting to note, considering the decrease training time for the model.
- Likewise, using dropouts does not change how RNNs are much more computationally expensive to train, compared to their CNN counterparts.
- Word-level Text Categorization seems to be better suited for both CNNs and RNNs, as evident from the better accuracies, although the training time is longer.

Question 6

We perform experiments and compare the accuracies and training time of all the experimented model variants in the table below. For ease of comparison, we also include the original RNN base models that all the experimental offshoots are derived from:

Experiment	Max Validation Acc	Mean Validation Acc	Time (seconds)
CharRNN	*0.7142*	*0.6769*	*2583*
WordRNN	*0.8614*	*0.8239*	*2492*
CharRNN with Dropout	0.7142	0.6567	2163
WordRNN with Dropout	0.8842	0.8657	2300
Vanilla Char RNN Layer	0.1428	0.1002	1181
Vanilla Word RNN Layer	0.0971	0.0735	1308
LSTM Char RNN Layer	0.6971	0.5978	2075
LSTM Word RNN Layer	0.7771	0.6752	2217
2-Layer Char RNN	0.7585	0.7164	4554
2-Layer Word RNN	0.9085	0.8896	4787
Gradient Clipping Char RNN	0.7071	0.6627	2731
Gradient Clipping Word RNN	0.8400	0.8062	2857

From the results tabled above, we postulate that for Text Classification using RNN Classifiers:

- The replacement of the GRU RNN Layer with a Vanilla RNN Layer results in very poor model performance (mean validation accuracy of ~0.1) for Char and Word RNNs.

- The replacement of the GRU RNN Layer with a LSTM RNN Layer results in a shorter time required for model training.
    - However, this also corresponds with decrease model performance. Hence, this is a trade-off that must be considered, depending on what metric is more important

- The use of a 2-Layer Char or Word RNN results in the best model performance.
    - However, this is counterbalanced by the prohibitively long time required to train (2x increase in training time for a 2-6% increase in performance, as compared to their normal RNN counterparts)

- The use of Gradient Clipping results in a longer time required for model training.
    - Unfortunately, it also results in a decrease in model performance. This would mean that gradient clipping serves no tangible benefit for RNNs, at least in this controlled scenario.